装饰器风格的Web服务框架

- 什么是装饰器
- 如何利用装饰器实现路由注册
- 如何利用装饰器实现中间件注册和数据校验
- 如何利用装饰器实现数据库整合

搭建TS环境

package.json创建

```
npm init -y
```

开发依赖安装

```
npm i typescript ts-node-dev tslint @types/node -D
```

设置启动脚本

```
"scripts": {
    "start": "ts-node-dev ./src/index.ts -P tsconfig.json --no-cache",
    "build": "tsc -P tsconfig.json && node ./dist/index.js",
    "tslint": "tslint --fix -p tsconfig.json"
}
```

加入tsconfig.json

```
{
    "compilerOptions": {
        "outDir": "./dist",
        "target": "es2017",
        "module": "commonjs",//组织代码方式
        "sourceMap": true,
        "moduleResolution": "node", // 模块解决策略
        "experimentalDecorators": true, // 开启装饰器定义
        "allowSyntheticDefaultImports": true, // 允许es6方式import
        "lib": ["es2015"],
        "typeRoots": ["./node_modules/@types"],
    },
    "include": ["src/**/*"]
}
```

创建入口文件

```
// ./src/index.ts
console.log('hello');
```

运行测试

```
npm start
```

装饰器是什么

概念介绍

> 装饰器模式（Decorator Pattern）允许向一个现有的对象添加新的功能，同时又不改变其结构。这种类型的设计模式属于结构型模式，它是作为现有的类的一个包装。
>
> 这种模式创建了一个装饰类，用来包装原有的类，并在保持类方法签名完整性的前提下，提供了额外的功能。
>
> 我们通过下面的实例来演示装饰器模式的用法。其中，我们将把一个形状装饰上不同的颜色，同时又不改变形状类。

定义一个方法

```
class Log {
    print(msg) {
        console.log(msg)
    }
}
const log = new Log()
log.print('hello')
```

装饰一下Print函数

- 日志美化
- 执行日志 AOP

  ```
  const dec = (target,property) => {
      const old = target.prototype.print
      target.prototype[property] = msg => {
          console.log('执行print方法...')
          msg = `{${msg}}`
          old(msg)
      }
  }
  dec(Log,'print')
  ```

装饰器工厂

- 打印定制化

```
const dec = name => (target,property) => {
    const old = target.prototype.print
    target.prototype[property] = msg => {
        console.log('执行print方法...')
        msg = `{${msg}} ${name}`
        old(msg)
    }
}
dec('Josephxia')(Log,'print')
```

注解风格的装饰器

```
function decorate(target, property, descriptor) {
    var oldValue = descriptor.value;
    descriptor.value = msg => {
        msg = `[${msg}]`
        return oldValue.apply(null, [msg]);
    }
    return descriptor;
}
class Log {
    @decorate
    print(msg) {
        console.log(msg)
    }
}
```

```

const anotation = (target,proterty,decorate) => {
    const descriptor = decorate(target.prototype, proterty,
Object.getOwnPropertyDescriptor(target.prototype, proterty))
    Object.defineProperty(target.prototype ,proterty,descriptor)
}
anotation(Log,'print',decorate)
```

搭建Koa环境

安装依赖

```
npm i koa koa-static koa-body koa-router @types/koa @types/koa-body @types/koa-
router -s
```

编写基础代码

index.ts

```
import * as Koa from 'koa'
import * as bodify from 'koa-body';
import * as Router from 'koa-router'
const app = new Koa()

app.use(
    bodify({
        multipart: true,
        // 使用非严格模式，解析 delete 请求的请求体
        strict: false,
    }),
);

const router = new Router()
router.get('/abc',ctx => {
    ctx.body = 'abc'
})
app.use(router.routes())

app.listen(3000, () => {
    console.log('服务器启动成功');
});
```

启动项目

```
npm start
```

路由定义及发现

创建路由

./src/routes/user.ts

```
import * as Koa from 'koa';

const users = [{ name: 'tom', age: 20 }, { name: 'tom', age: 20 }];
export default class User {
    @get('/users')
    public list(ctx: Koa.Context) {
        ctx.body = { ok: 1, data: users };
    }

    @post('/users')
    public add(ctx: Koa.Context) {
        users.push(ctx.request.body);
        ctx.body = { ok: 1 }
    }

}
```

> 知识点补充：装饰器的编写，以@get('/users')为例，它是函数装饰器且有配置项，其函数签名为：

> ```
> function get(path) {
> return function(target, property, descriptor) {}
> }
> ```

> 另外需解决两个问题：
>
> 1. 路由发现
> 2. 路由注册

1. 路由发现及注册，创建./utils/route-decors.ts

   ```
   npm i glob @types/glob -s
   ```

   ```
   import * as glob from 'glob';
   import * as Koa from 'koa';
   import * as KoaRouter from 'koa-router';

   const router = new KoaRouter()
   export const get = (path: string) => {
       return (target, property) => {
           router['get'](path, target[property])
       }
   }

   export const post = (path: string) => {
       return (target, property) => {
           router['post'](path, target[property])
       }
   }
   ```

   解决get post put delete方法公用逻辑

   需要进一步对原有函数进行柯里化

   ```
   const router = new KoaRouter()

   const method = method => (path: string, options?: RouteOptions) => {
       return (target, property, descriptor) => {
           const url = options && options.prefix ? options.prefix + path : path
           router[method](url, target[property])
       }
   }
   export const get = method('get')
   export const post = method('post')
   ```

   router变量 不符合函数式编程引用透明的特点 对后面移植不利

   所以要再次进行柯里化

   ```
   import * as glob from 'glob';
   import * as Koa from 'koa';
   import * as KoaRouter from 'koa-router';
   ```

```
const router = new KoaRouter()
const method = (router:KoaRouter) => (method: 'get' | 'post' | 'delete' |
'put') => (path: string) => {
    return (target, property) => {
        router[method](path, target[property])
    }
}

const decorate = method(router)
export const get = decorate('get')
export const post = decorate('post')
```

```
export const load = (folder: string): KoaRouter => {
    const extname = '.{js,ts}'
    glob.sync(require('path')
        .join(folder, `./**/*${extname}`))
        .forEach((item) => require(item))
    return router
}
```

2. 使用

   routes/user.ts

```
import { get, post } from '../utils/decors'
```

   index.ts

```
import { load } from './utils/decors';
import {resolve} from 'path'
const router = load(resolve(__dirname, './routes'));
app.use(router.routes())
```

3. 数据校验：可以利用中间件机制实现

   添加校验函数，./routes/user.ts

```
export default class User {
    // 添加中间件选项
    @post('/users', {
        middlewares: [
            async function validation(ctx: Koa.Context, next: () =>
Promise<any>) {
                // 用户名必填
                const name = ctx.request.body.name
                if (!name) {
                    throw "请输入用户名";
                }
              await next();
            }
        ]
    })
    public async add(ctx: Koa.Context) {}
}
```

更新decors.ts

```
export const load = function(prefix: string, folder: string, options:
LoadOptions = {}): KoaRouter {
        // ...
    route = function(method: HTTPMethod, path: string, options? : {middlewares:
Array<any>} ) {
        return function(target, property: string, descriptor) {
            // 添加中间件数组
           const middlewares = [];

           // 若设置了中间件选项则加入到中间件数组
           if (options.middlewares) {
               middlewares.push(...options.middlewares);
           }

           // 添加路由处理器
           middlewares.push(target[property]);
           // router[method](url, target[property]);
           router[method](path, ...middlewares);
        };
    };

    // ...
    return router;
};
```

5. 类级别路由守卫

使用，routes/user.ts

```
@middlewares([
    async function guard(ctx: Koa.Context, next: () => Promise<any>){
        console.log('guard', ctx.header);

        if(ctx.header.token) {
            await next();
        } else {
            throw "请登录";
        }
    }
])
export default class User {}
```

增加中间装饰器，更新route-decors.ts

```
//增加中间装饰器
export const middlewares = function middlewares(middlewares:
Koa.Middleware[]) {
    return function(target) {
        target.prototype.middlewares = middlewares;
    };
};

//修改load方法
export const load = function(prefix: string, folder: string, options:
LoadOptions = {}): KoaRouter {

    route = function(method: HTTPMethod, path: string, options: RouteOptions
= {}) {
        return function(target, property: string, descriptor) {
            // 晚一拍执行路由注册：因为需要等类装饰器执行完毕
            process.nextTick(() => {
                let mws = [];
                // 获取class上定义的中间件
                if (target.middlewares) {
                    middlewares.push(...target.middlewares);
                }
                // ...
            });
        };
    };

    return router;
};
```

数据库整合

安装依赖

npm i -S sequelize sequelize-typescript reflect-metadata mysql2`

初始化

npm i sequelize-typescript@0.6.11

npm i sequelize@5.8.12

index.ts

```
    import { Sequelize } from 'sequelize-typescript';

    const database = new Sequelize({
        port:3306,
        database:'kaikeba',
        username:'root',
        password:'example',
        dialect:'mysql',
        modelPaths: [`${__dirname}/model`],
    });
    database.sync({force: true})
```

创建模型

```
// model/user.js
import { Table, Column, Model, DataType } from 'sequelize-typescript';

@Table({modelName: 'users'})
export default class User extends Model<User> {
    @Column({
        primaryKey: true,
        autoIncrement: true,
        type: DataType.INTEGER,
    })
    public id: number;

    @Column(DataType.CHAR)
    public name: string;
}
```

使用模型

routes/user.ts

```
import model from '../model/user';

export default class User {

    @get('/users')
    public async list(ctx: Koa.Context) {
        const users = await model.findAll()
        ctx.body = { ok: 1, data: users };
    }
}
```

框架不足

- Restful接口
- model可以自动加载到ctx中
- service层自动加载